Once the data object is no longer reachable by a mutator, object termination begins. Typically, a garbage collector determines reachability using any of a variety of suitable techniques; however, explicit reclamation techniques may be employed in some realizations to trigger object termination. During this termination process, the object sampler collects additional information about the data object such as its termination time, and the weak reference established by the object sampler is removed. The object sampler then compiles and updates data object lifetime statistics based in part on the newly collected information. In some realizations, object statistics may be updated apart from termination. For example, in a generational collector implementation, statistics suitable for shaping a tenuring policy may be updated based on populations of sampled objects promoted from a younger generation to an older generation.

3. Please delete paragraph [1056] beginning on page 18, line 10, and substitute therefor the following paragraph:

[1056] Other possible selection mechanisms include:

4. Counter: A counter can be employed as an arbitrary selection mechanism. For example, in one realization, a counter initialized with some positive value is decremented each time an object is allocated. Selection employs a computationally efficient triggering mechanism. For example, if the carry-bit of the decrement is added into the address of the free pointer, then when the counter underflows, the load from the free pointer will be biased causing a misalignment trap. The trap handler can either perform the sampling directly or patch the allocation site to sample the next allocated object. Such an approach imposes some additional computational load (e.g., 4 extra instructions in the fast-path), but avoids skewing.

5. Other overflow: One variation is to build on the basic LAB overflow strategy (described above) but use a smaller limit than actual size of LAB to provide more frequent sampling. The limit can be varied within a range to reduce skew. This variation may be particularly useful in implementations for which the LAB size is large.

6. Sampling functions: Another approach is to use sampling functions such as the pcsample(2) system call provided in the Solaris™ operating system to gather a set of hot program instructions. In general, we can use such functions to identify hot methods or to build hot (possibly interprocedural) paths. By patching or recompiling all allocation sites in these hot areas we can provide selective

- 3 -

PATENT

sampling. The idea is that objects allocated together are likely to have similar lifetimes and should be studied as a group.

7. Combine with static analysis: If we statically determine that a set of allocation-sites allocate objects that are referenced from a sampled allocation-site (e.g., a site selected using one of the selection methods described above), then we can apply a tenuring decision for that sampled site to the referenced set.

3. Please delete paragraph [1076] beginning on page 26, line 2, and substitute therefor the following paragraph:

[1076] There are a number of ways in which dynamically sampled object lifespan information may be used to improve the placement of objects. For example, such information may be employed on allocation of corresponding objects, when promoting an object to an older generation, or when moving an object within a generation. In each case, a decision to place a particular instance of an object may be informed by lifetime information sampled for a corresponding sampled set of objects. For example, sampled object lifetime information may be employed to facilitate pretenuring in automatic dynamic memory management such as described herein or as more completely described in the co-pending U.S. Patent Application No. 09/855,453, entitled "Dynamic Adaptive Pretenuring of Objects," naming Agesen, Garthwaite and Harris as inventors and filed May 15, 2001, the entirety of which is hereby incorporated by reference. Similarly, promotion of objects allocated at a given site may be accelerated if sampled object lifetimes suggest that corresponding objects exhibit bimodal behavior—for example, either dying young or living forever. Accordingly, a generational collector exploiting such lifetime information may promote those objects that survive a dynamically-calculated lifetime threshold directly to the oldest generation.

4. Please delete paragraph [1077] beginning on page 26, line 19, and substitute therefor the following paragraph:

[1077] In the case of adaptive allocation decisions such as pretenuring, fast path allocation efficiency may be improved if we are able to recompile the methods for the allocation sites where we wish to set a particular tenuring policy. In some execution environments such adaptation is straightforward. For example, some Java virtual machine implementations provide mechanisms for scheduling a method to be recompiled. When a change in policy is desired, the methods dependent on a given allocation-site are recompiled to implement the new object-placement policy. One advantage of implementations that provide some level of hysteresis in

- 4 -